Becker & Hickl GmbH
Nahmitzer Damm 30
12277 Berlin
Tel. +49 30 787 56 32
Fax. +49 30 787 57 34
email: info@becker-hickl.de
http://www.becker-hickl.de

LIA-150 32 Bit Dynamic Link Libraries

User Manual

Version 1.2

Dec. 99

Introduction

The LIA 32 bits Dynamic Link Library contains all functions to control the LIA-150 modules. The functions work under Windows 95 and Windows NT. The program which calls the DLLs must be compiled with the compiler option 'Structure Alignment' set to '1 Byte'.

The distribution disks contain the following files:

LIADLL32.DLL	dynamic link library main file
LIADLL32.LIB	import library file for Microsoft Visual C/C++, Borland C/C++, Watcom C/C++ and Symantec C/C++ compilers
LIA_DEF.H	Include file containing Types definitions, Functions Prototypes and Pre-processor statements
LIA150.INI	LIA DLL initialisation file
LIADLL32.DOC	This description file
LIADLL32.TXT	This description saved as a text file
USE_LIA.XXX	Set of files to compile and run a simple example of using the LIA DLL functions. The source file of the example is the file use_lia.c. The example was originally prepared under Borland C/C++, V.4.52. For use with other compilers, please choose the correct import library file to link.

There is no special installation procedure required. Simply execute the setup program from the 1st distribution diskette and follow its instructions.

LIA-DLL Functions list

The following functions are implemented in the LIA-DLL:

Initialisation functions:

LIA_init
LIA_get_id
LIA_test_id
LIA_test_if_active
LIA_get_init_status
LIA_set_simul
LIA_get_mode

Setup functions:

LIA_get_parameters
LIA_set_hard_parameters
LIA_set_range
LIA_get_range
LIA_set_lfilter
LIA_get_lfilter
LIA_set_hfilter
LIA_get_hfilter
LIA_set_dyn_reserve
LIA_get_dyn_reserve

LIA_set_phase_mode
LIA_get_phase_mode
LIA_set_ref_source
LIA_get_ref_source
LIA_set_ref_threshold
LIA_get_ref_threshold
LIA_get_ref_freq
LIA_get_ref_freq_range
LIA_set_harmonic
LIA_get_harmonic
LIA_set_trig_mode
LIA_get_trig_mode
LIA_set_out_filter
LIA_get_out_filter
LIA_set_out_rate
LIA_get_out_rate
LIA_set_offset
LIA_get_offset
LIA_set_dac
LIA_get_dac
LIA_get_eeprom_data
LIA_write_eeprom_data
LIA_get_adjust_parameters
LIA_set_adjust_parameters

Measurement control functions:

LIA_start_measure
LIA_stop_measure
LIA_read_sample

Status functions:

LIA_test_if_sample_ready
LIA_get_status

Application Guide

Initialisation of the LIA Measurement Parameters

Before a measurement is started the measurement parameter values must be written into the internal structures of the DLL functions (not directly visible from the user program) and sent to the control registers of the LIA module. This is accomplished by the function **LIA_init**.

The LIA DLL Functions can control up to four LIA modules.

The **LIA_init** function

- reads the parameter values from a specified file
- checks the base I/O addresses for all active modules to avoid hardware conflicts
- checks and recalculates the parameters depending on hardware restrictions and the adjust parameters from the EEPROM on each active LIA module
- sends the parameter values to the LIA control registers of each active LIA module
- performs a hardware test of each active LIA module

The initialisation file is an ASCII file with a structure shown in the table below. Each module has its own section in the initialisation file ([lia_module0..3]). Only modules which have an entry ‘ active = 1’ are initialised. We recommend either to use the file lia150.ini for initialisation or to start with lia150.ini and to introduce the desired changes.

```
;    LIA150 initialisation file
;    LIA parameters have to be included in .ini file only when parameter
;    value is different from default.
;    module section (lia_module0-3) is required for each existing LIA module
[lia_base]
hardware = 1                    ; 1 - hardware mode ,0 - simulation mode (default=1)
out_filter = 52.4               ;output filter setting [Hz] (default 52.4 )
out_rate = 200.                 ;output rate setting [samples/sec] (default 200.)
  ; see module documentation for possible values of out_filter and out_rate

[lia_module0]
baseadr=  0x380                 ;base I/O address   (0 ... 0x3FC,default 0x380)
active = 1                      ;module active - can be used  (default = 0 - not active)
range = 0                       ;input signal range  0 (default) .. 7
                                ;      100, 50, 20, 10, 5, 2, 1, 0.5 mV
lfilter = 0                     ;low pass input filter  0 (default) ... 5
                                ;      3MHz(off), 1MHz, 100kHz, 10kHz, 1kHz, 100Hz
hfilter = 0                     ;high pass input filter  0(default) ... 3
                                ;      1Hz(off), 10Hz, 100Hz, 1kHz
dyn_reserve = 0                 ; dynamic reserve  0-low (default),1-medium ,2-high
phase_mode  = 0                 ; phase mode   0-dual phase (default),1- single phase
ref_source  = 1                 ; reference signal source :
                                ;  1 - external (default) , 0 - internal (from input signal)
ref_threshold = 1.0             ;reference signal threshold [V]
                                ;     (-2.5V .. +2.5V , default 1.0 V )
harmonic = 1                    ;output signal is : 1 -1st harmonic (default) 2 - 2nd harmonic
trig_mode = 0                   ; trigger mode : 0(default) - none, 1 - Low ,2 - High
offset_A = 0                    ; channel A offset (-10000 .. +10000 , default 0  )
offset_B = 0                    ; channel B offset (-10000 .. +10000 , default 0  )

[lia_module1]

baseadr = 0x280                 ;base I/O address   (0 ... 0x3FC,default 0x280)
active = 0                      ;module not active - can't  be used

[lia_module2]

baseadr = 0x2a0                 ;base I/O address   (0 ... 0x3FC,default 0x2a0)
active = 0                      ;module not active - can't  be used

[lia_module3]

baseadr= 0x2c0                  ;base I/O address   (0 ... 0x3FC,default 0x2c0)
active = 0                      ;module not active - can't  be used
```

After a **LIA_init** call we recommend to check which PMS modules are active by calling **LIA_test_if_active** function. At least one module must be active, and only active modules can be operated further. It is recommended (but not required) to check also the initialisation status (**LIA_get_init_status**) of each used module. In case of a wrong initialisation the initialisation status shows the reason of the error (see lia_def.h for possible values).

After calling the **LIA_init** function the measurement parameters from the initialisation file are present in the module control registers and in the internal data structures of the DLLs. To

give the user access to the parameters, the function **LIA_get_parameters** is provided. This function transfers the parameter values from the internal structures of the DLLs into a structure of the type LIAdata (see lia_def.h) which has to be defined by the user. The parameter values in this structure are described below.

unsigned short base_adr	base I/O address (0...0x3E0,default 0x380)
short error;	last error code
short enable_meas;	enable/disable(1/0) measurement
short init;	set to initialisation result code
short active;	most of the library functions are executed only when module is active (not 0)
unsigned short range;	input signal range 0 .. 7 , (100, 50, 20, 10, 5, 2, 1, 0.5mV)
unsigned short lfilter;	low pass input filter 0 .. 5 , (3MHz(off), 1MHz, 100kHz, 10kHz, 1kHz, 100Hz)
unsigned short hfilter;	high pass input filter 0 .. 3 , (1Hz (off), 10, 100 Hz, 1 kHz
unsigned short dyn_reserve;	Dynamic Reserve 0-low ,1-medium ,2-high
unsigned short phase_mode ;	0 - Dual Phase ,1 - Single Phase
unsigned short ref_source ;	reference signal source : 1 - external , 0 - internal (from input signal)
float ref_threshold;	reference signal threshold [V]
float ref_freq;	reference signal frequency [Hz]
unsigned short harmonic;	output signal is : 1 (2)- 1st(2nd) harmonic
unsigned short status;	bit 2 - PLL state 0 - Locked , 1 - Unlocked bit 1 - input signal overload bit 0 - triggered (reference clock present)
float out_filter;	output filter setting [Hz] (0.00524 .. 52.4)
float out_rate;	output rate setting [samples/sec]
unsigned short trig_mode;	trigger mode : 0 - none, 1 - Low ,2 - High
float dacout1;	DAC voltage of DACout1
float dacout2;	DAC voltage of DACout2
unsigned long adc_A;	last read value from ADC in channel A
unsigned long adc_B;	last read value from ADC in channel B
long offset_A;	channel A offset +-10000
long offset_B;	channel B offset +-10000

To send the hardware parameters back to the DLLs and to a LIA module (e.g. after changing parameter values) the function **LIA_set_hard_parameters** is used. This function checks and - if required - recalculates all parameter values due to cross dependencies and hardware restrictions. Therefore, it is recommended to read the parameter values after calling LIA_set_hard_parameters by LIA_get_parameters.

Single parameter values can be transferred to or from the DLL and module level by the proper function (e.g. for input signal range use **LIA_set_range** and **LIA_get_range).**

Standard Measurements

The most important measurement functions are listed below.

LIA_start_measure initialises the measurement with the parameters set before by LIA_init or other parameter setting functions. The procedure initialises the DLL internal variables, performs an auto-calibration and enables reading ADC values. If the test_trigger input

parameter is ‘1’, the procedure at the beginning tests whether a sweep trigger event occurred in all active LIA modules.

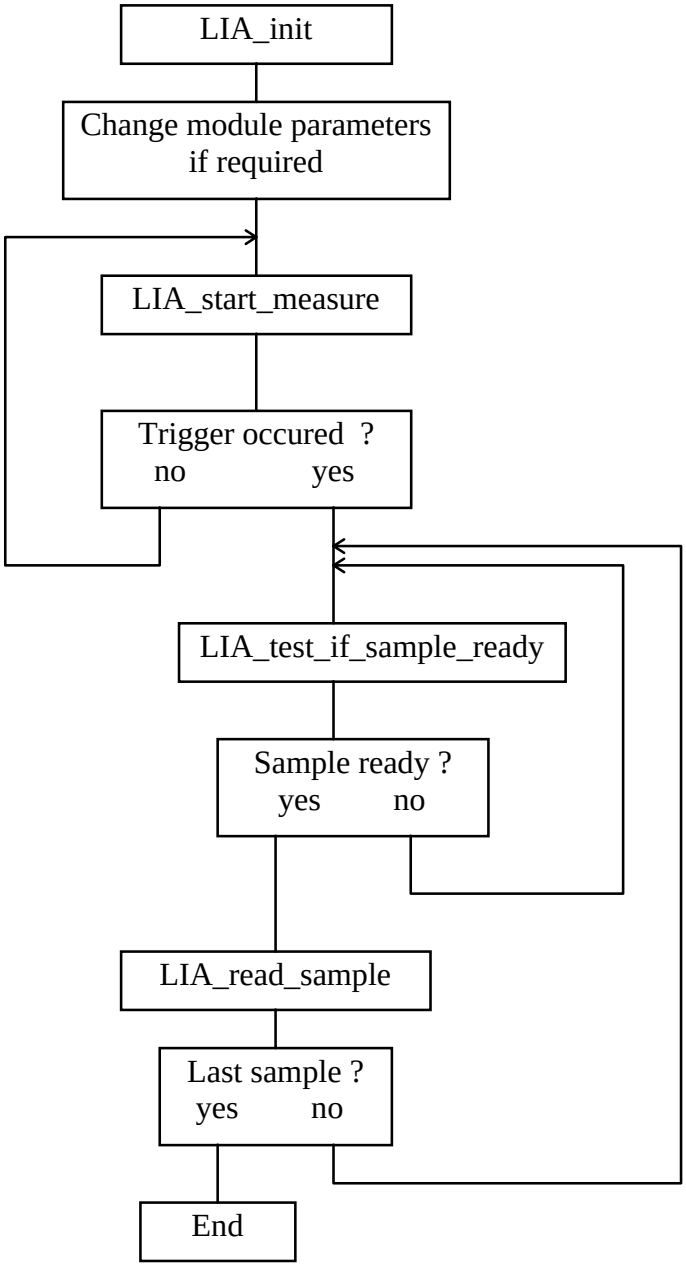
LIA_stop_measure is used to stop the measurement by a software command. It disables reading ADC values on all LIA modules.

LIA_test_if_sample_ready is used to check whether a new sample value is available from all active LIA modules.

LIA_get_status is used to test the state of a LIA module. The status bits delivered by the function are listed below.

0x1	bit 0 - triggered (reference clock present)
0x2	bit 1 - input signal overload
0x4	bit 2 - PLL state 0 - Locked , 1 - Unlocked

A simple measurement sequence is shown in the block diagram below.



Error Handling

Each LIA DLL function returns an error status. Return values >= 0 indicate error free execution. A value < 0 indicates that an error occurred during execution. The meaning of a particular error code can be found in lia_def.h file. We recommend to check the return value after each function call.

Description of the LIA DLL Functions

---

short LIA_DECL **LIA_init** (char far config_file[],short far *hndl_table);

Input parameters:

config_file	pointer to a string containing the name of the initialisation file in use (including file name and extension)
*hndl_table	pointer to a table which will be filled with module handles

Return value:

0 no errors, <0 error code

Description:

Before a measurement is started the measurement parameter values must be written into the internal structures of the DLL functions (not directly visible from the user program) and sent to the control registers of the LIA module. This is accomplished by the function **LIA_init**.

The function at the beginning fills hndl_table with handles which are next used to call the module specific functions. A handle value >= 0 means that the function successfully created the internal DLL structures for the module. The hndl_table must have space for 4 entries (max. no of LIA module).

The LIA_init function

- reads the parameter values from the specified file config_file
- checks sync_adr and base I/O addresses for all active modules to avoid hardware conflicts
- checks and recalculates the parameters depending on hardware restrictions and adjust parameters from the EEPROM in each active LIA module
- sends the parameter values to the internal structures of the DLL
- sends the parameter values to the LIA control registers of each active LIA module
- performs a hardware test of each active LIA module

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file lia150.ini or to start with lia150.ini and introduce the desired changes.

```
;   LIA150 initialisation file
;   LIA parameters have to be included in .ini file only when parameter
;   value is different from default.
;   module section (lia_module0-3) is required for each existing LIA module
[lia_base]
hardware = 1                    ; 1 - hardware mode ,0 - simulation mode (default=1)
out_filter = 52.4               ;output filter setting [Hz] (default 52.4 )
out_rate = 200.                 ;output rate setting [samples/sec] (default 200.)
  ; see module documentation for possible values of out_filter and out_rate

[lia_module0]
baseadr=  0x380                 ;base I/O address   (0 ... 0x3FC,default 0x380)
active = 1                      ;module active - can be used  (default = 0 - not active)
range = 0                       ;input signal range  0 (default) .. 7
                                ;      100, 50, 20, 10, 5, 2, 1, 0.5 mV
```

```
lfilter = 0                 ;low pass input filter  0 (default) ... 5
                            ;      3MHz(off), 1MHz, 100kHz, 10kHz, 1kHz, 100Hz
hfilter = 0                 ;high pass input filter  0(default) ... 3
                            ;      1Hz(off), 10Hz, 100Hz, 1kHz
dyn_reserve = 0             ; dynamic reserve  0-low (default),1-medium ,2-high
phase_mode  = 0             ; phase mode   0-dual phase (default),1- single phase
ref_source  = 1             ; reference signal source :
                            ;  1 - external (default) , 0 - internal (from input signal)
ref_threshold = 1.0         ;reference signal threshold [V]
                            ;     (-2.5V .. +2.5V , default 1.0 V )
harmonic = 1                ;output signal is : 1 -1st harmonic (default) 2 - 2nd harmonic
trig_mode = 0               ; trigger mode : 0(default) - none, 1 - Low ,2 - High
offset_A = 0                ; channel A offset (-10000 .. +10000 , default 0  )
offset_B = 0                ; channel B offset (-10000 .. +10000 , default 0  )

[lia_module1]

baseadr = 0x280             ;base I/O address   (0 ... 0x3FC,default 0x280)
active = 0                  ;module not active - can't  be used

[lia_module2]

baseadr = 0x2a0             ;base I/O address   (0 ... 0x3FC,default 0x2a0)
active = 0                  ;module not active - can't  be used

[lia_module3]

baseadr= 0x2c0              ;base I/O address   (0 ... 0x3FC,default 0x2c0)
active = 0                  ;module not active - can't  be used
```

After **LIA_init** call we recommend to check which LIA modules are active by calling the **LIA_test_if_active** function. At least one module must be active, and only active modules can be operated further. It is reasonable to check also the initialisation status (**LIA_get_init_status**) of each module used. The initialisation status can show the reason of a wrong initialisation (see lia_def.h for possible values).

---

short LIA_DECL **LIA_get_id**(short hndl, short far* id);

Input parameters:

hndl	module handle
*id	pointer to the identification code

Return value:

0 no errors, <0 error code

Description:

The procedure loads the ‘id’ variable with the identification code of the LIA module ‘hndl’. LIA_get_id is a low level procedure which is called also during the initialisation by LIA_init.

---

short LIA_DECL **LIA_test_id**(short hndl);

Input parameters:

hndl	module handle

Return value:

0 correct id , <0 error code

Description:

The procedure checks whether the LIA module ‘hndl’ has the correct identification code. LIA_test_id is a low level procedure called already during the initialisation by LIA_init.

---

short LIA_DECL **LIA_test_if_active** (short hndl);

Input parameters:

hndl	module handle

Return value:

0 - module not active (cannot be used) , 1 - module active

Description:

The procedure returns information whether the module specified by ‘hndl’ is active or not. A module is set active only if there is the entry ‘active = 1’ in the respective module section in the config_file (see LIA_init function). As a result of a wrong initialisation (LIA_init function) a module can be deactivated. In this case the LIA_get_init_status function (see below) is recommendecd to find the reason of deactivating the module.

---

short LIA_DECL **LIA_get_init_status**(short hndl, short far* ini_status);

Input parameters:

hndl	module handle
*ini_status	pointer to the initialisation status

Return value:

0 no errors, <0 error code

Description:

The procedure the loads ini_status variable with the initialisation result code set by the function LIA_init for module ‘hndl’. The possible values are shown below (see also lia_def.h):

INIT_OK	0	no error
INIT_WRONG_MOD_ID	-1	wrong module identification code
INIT_WRONG_EEP_CHKSUM	-2	wrong EEPROM checksum
INIT_WRONG_DACS	-3	DAC’s test failed
INIT_NOT_DONE	-4	init. not done

---

short LIA_DECL **LIA_set_simul**(void);

Input parameters:

none

Return value:

0 no errors , <0 error code

Description:

The procedure is used to change the DLL operating mode to ‘simulation’. This mode is used to run the software without LIA modules or in case of hardware errors found during the LIA_init call.

---

short LIA_DECL **LIA_get_mode**(void);

Input parameters:

none

Return value:

0 simulation mode , 1 hardware mode

Description:

The procedure returns current DLL operating mode.

---

short LIA_DECL **LIA_get_parameters**(short hndl, LIAdata far * data);

Input parameters:

hndl	module handle
data	pointer to result structure (type LIAdata)

Return value: 0

Description:

After calling the **LIA_init** function (see above) the measurement parameters from the initialisation file are present in the module and in the internal data structures of the DLLs. To give the user access to the parameters, the function **LIA_get_parameters** is provided. This function transfers the parameter values of the module 'hndl' from the internal structures of the DLLs into a structure of the type LIAdata (see lia_def.h) which has to be defined by the user. The parameter values in this structure are described below.

unsigned short base_adr	base I/O address (0...0x3E0,default 0x380)
short error;	last error code
short enable_meas;	enable/disable(1/0) measurement
short init;	set to initialisation result code
short active;	most of the library functions are executed only when module is active (not 0)
unsigned short range;	input signal range 0 .. 7 , (100, 50, 20, 10, 5, 2, 1, 0.5mV)
unsigned short lfilter;	low pass input filter 0 .. 5 , (3MHz(off), 1MHz, 100kHz, 10kHz, 1kHz, 100Hz)
unsigned short hfilter;	high pass input filter 0 .. 3 , (1Hz (off), 10, 100 Hz, 1 kHz
unsigned short dyn_reserve;	Dynamic Reserve 0-low ,1-medium ,2-high
unsigned short phase_mode ;	0 - Dual Phase ,1 - Single Phase
unsigned short ref_source ;	reference signal source : 1 - external , 0 - internal (from input signal)
float ref_threshold;	reference signal threshold [V]
float ref_freq;	reference signal frequency [Hz]
unsigned short harmonic;	output signal is : 1 (2)- 1st(2nd) harmonic
unsigned short status;	bit 2 - PLL state 0 - Locked , 1 - Unlocked bit 1 - input signal overload bit 0 - triggered (reference clock present)
float out_filter;	output filter setting [Hz] (0.00524 .. 52.4)
float out_rate;	output rate setting [samples/sec]
unsigned short trig_mode;	trigger mode : 0 - none, 1 - Low ,2 - High
float dacout1;	DAC voltage of DACout1
float dacout2;	DAC voltage of DACout2
unsigned long adc_A;	last read value from ADC in channel A
unsigned long adc_B;	last read value from ADC in channel B
long offset_A;	channel A offset +-10000
long offset_B;	channel B offset +-10000

---

short LIA_DECL **LIA_set_hard_parameters**(short hndl, LIAdata far *data);

Input parameters:

hndl	module handle
data	pointer to result structure (type LIAdata)

Return value:
0 no errors , <0 error code

Description:

The procedure sends all parameters from the ‘data’ structure to the internal DLL structures and to the control registers of the LIA module.

The new parameter values are recalculated according to the parameter limits, hardware restrictions (e.g. DAC resolution). Furthermore, cross dependencies between different parameters are taken into account to ensure the correct hardware operation. It is recommended to read back the parameters after setting to get their real values after recalculation.

The values of ‘base_adr’, ‘init’ and ‘active’ are not changed. They can be changed only by a new config_file and a new LIA_init call.

If an error occurs at a particular parameter, the procedure does not set the rest of the parameters and returns with an error code.

---

short LIA_DECL **LIA_set_range**(short hndl, short range);

Input parameters:

hndl	module handle
range	new range value- 0 .. 7 (for 100, 50, 20, 10, 5, 2, 1, 0.5 mV)

Return value:
0 no errors , <0 error code

Description:

The procedure sets the input signal range for the LIA module ‘hndl’.

---

short LIA_DECL **LIA_get_range**(short hndl, short far * range);

Input parameters:

hndl	module handle
range	pointer to result value

Return value:
0 no errors , <0 error code

Description:

The procedure loads the ‘range’ variable with the actual value of the input signal range parameter for the LIA module ‘hndl’. The ‘range’ value is in the range of 0 .. 7, which means 100, 50, 20, 10, 5, 2, 1, 0.5 mV.

---

short LIA_DECL **LIA_set_lfilter**(short hndl, short lfilter);

Input parameters:

hndl	module handle
lfilter	new low pass input filter value - 0 .. 5 (for 3MHz(off), 1MHz, 100kHz, 10kHz, 1kHz, 100Hz)

Return value:
0 no errors , <0 error code

Description:

The procedure sets the low pass input filter parameter value for the LIA module ‘hndl’.

---

short LIA_DECL **LIA_get_lfilter**(short hndl, short far * lfilter);

Input parameters:

hndl	module handle
lfilter	pointer to result value

Return value:
0 no errors , <0 error code

Description:

The procedure loads the‘lfilter’ variable with the actual value of the low pass input filter setting for the LIA module ‘hndl’. The low pass input filter setting is in the range of 0 .. 5, which means (3MHz(off), 1MHz, 100kHz, 10kHz, 1kHz, 100Hz).

---

short LIA_DECL **LIA_set_hfilter**(short hndl, short hfilter);

Input parameters:

hndl	module handle

hfilter	new high pass input filter value - 0 .. 3 (for 1Hz(off), 10Hz, 100Hz, 1kHz)

Return value:
0 no errors , <0 error code

Description:

The procedure sets the high pass input filter of the LIA module ‘hndl’.

---

short LIA_DECL **LIA_get_hfilter**(short hndl, short far * hfilter);

Input parameters:

hndl	module handle
hfilter	pointer to result value

Return value:
0 no errors , <0 error code

Description:

The procedure loads the ‘hfilter’ variable with the actual value of the high pass input filter setting for the LIA module ‘hndl’. The setting is in the range of 0 .. 3, which means 1Hz(off), 10Hz, 100Hz, and 1kHz.

---

short LIA_DECL **LIA_set_dyn_reserve**(short hndl, short dyn_reserve);

Input parameters:

hndl	module handle
dyn_reserve	new dynamic reserve 0-low ,1-medium ,2-high

Return value:
0 no errors , <0 error code

Description:

The procedure sets the dynamic reserve parameter value for the LIA module ‘hndl’.

---

short LIA_DECL **LIA_get_dyn_reserve**(short hndl, short far * dyn_reserve);

Input parameters:

hndl	module handle
dyn_reserve	pointer to result value

Return value:
0 no errors , <0 error code

Description:

The procedure loads the ‘dyn_reserve’ variable with the actual value of the dynamic reserve setting for the LIA module ‘hndl’. The dynamic reserve setting can be 0-low ,1-medium or 2-high.

---

short LIA_DECL **LIA_set_phase_mode**(short hndl, short phase_mode);

Input parameters:

hndl	module handle
phase_mode	new phase mode, 0-dual phase, 1- single phase

Return value:
0 no errors , <0 error code

Description:

The procedure sets the phase mode parameter value for the LIA module ‘hndl’.

---

short LIA_DECL **LIA_get_phase_mode**(short hndl, short far * phase_mode);

Input parameters:

hndl	module handle
phase_mode	pointer to result value

Return value:
0 no errors , <0 error code

Description:

The procedure loads the ‘phase_mode’ variable with the actual value of the phase mode setting for the LIA module ‘hndl’. The phase mode setting can be 0 - dual phase or 1 - single phase.

---

short LIA_DECL **LIA_set_ref_source**(short hndl, short ref_source);

Input parameters:

hndl	module handle
ref_source	new reference signal source, 1 - external (default) , 0 - internal (from the input signal)

Return value:
0 no errors , <0 error code

Description:

The procedure sets the reference signal source for the LIA module ‘hndl’.

---

short LIA_DECL **LIA_get_ref_source**(short hndl, short far * ref_source);

Input parameters:

hndl	module handle
ref_source	pointer to result value

Return value:
0 no errors , <0 error code

Description:

The procedure loads the ‘ref_source’ variable with the actual value of the reference signal source setting for the LIA module ‘hndl’. Reference signal source setting can be 1 - external (default) or 0 - internal (from the input signal).

---

short LIA_DECL **LIA_set_ref_threshold**(short hndl, float threshold);

Input parameters:

hndl	module handle
threshold	Reference Threshold

Return value:
0 no errors , <0 error code

Description:

The procedure sets the reference threshold parameter value for the LIA module ‘hndl’. ‘Threshold’ can be in the range from -2V to +2V.

---

short LIA_DECL **LIA_get_ref_threshold**(short hndl, float fat * threshold);

Input parameters:

hndl	module handle
threshold	pointer to result value

Return value:
0 no errors , <0 error code

Description:

The procedure loads the ‘threshold’ variable with the actual value of the reference threshold for the LIA module ‘hndl’.

---

short LIA_DECL **LIA_get_ref_freq**(short hndl, float far * ref_freq);

Input parameters:

hndl	module handle
ref_freq	pointer to result value

Return value:
0 no errors , 1 - frequency value invalid, <0 error code

Description:

The procedure loads the ‘ref_freq’ variable with the actual value of the reference frequency for the LIA module ‘hndl’. The reference frequency is measured by a counter in the LIA module which is read in intervals determined by the actual ‘frequency range’. Therefore, LIA_get_ref_freq can deliver invalid results (e.g. immediately after a initialisation). Such situations are indicated by a ‘1’ in the return value.

---

short LIA_DECL **LIA_get_ref_freq_range**(short hndl, float far * range);

Input parameters:

hndl	module handle
range	pointer to result value

Return value:
0 no errors , <0 error code

Description:

The procedure loads the ‘range’ variable with the actual value of the reference frequency range for the LIA module ‘hndl’. The reference frequency range is set automatically in the LIA module(s) according to the last reference frequency measurement. It is used to select the appropriate range of the PLL. All these actions occur automatically in the LIA module. Therefore, LIA_get_ref_freq_range is required only for test purposes or to give additional information about the module function.

---

short LIA_DECL **LIA_set_harmonic**(short hndl, short harmonic);

Input parameters:

hndl	module handle
harmonic	new harmonic setting, 1 -1st harmonic, 2 - 2nd harmonic

Return value:
0 no errors , <0 error code

Description:

Controlled by the value of ‘harmonic’ the PLL of the LIA-150 locks either on the reference signal or on the 2nd harmonic of the reference signal. The LIA module to which the setting applies is determined by ‘hndl’.

---

short LIA_DECL **LIA_get_harmonic**(short hndl, short far * harmonic);

Input parameters:

hndl	module handle
harmonic	pointer to result value

Return value:
0 no errors , <0 error code

Description:

The procedure loads the ‘harmonic’ variable with the actual value of the output signal harmonic setting for the LIA module ‘hndl’. The result can be 1 (1st harmonic), or 2 (2nd harmonic).

---

short LIA_DECL **LIA_set_trig_mode**(short hndl, short trig_mode);

Input parameters:

hndl	module handle
trig_mode	new trigger mode 0 - none, 1 - Low ,2 - High

Return value:

0 no errors , <0 error code

Description:

The procedure sets the sweep trigger mode for the LIA module ‘hndl’. For ‘0’ the recording starts immediately with the start of the measurement. For ‘1’ and ‘2’ the recording starts when the sweep trigger signal (at the sub-D connector, see LIA-150 description) assumes the sepcified TTL level.

---

short LIA_DECL **LIA_get_trig_mode**(short hndl, short far * trig_mode);

Input parameters:

hndl	module handle
trig_mode	pointer to result value

Return value:

0 no errors , <0 error code

Description:

The procedure loads ‘trig_mode’ variable with the actual value of the sweep trigger mode setting for the LIA module ‘hndl’. The result can be 0 (no sweep trigger, immediate start), 1 (start at ‘Low’), 2 (start at ‘High’).

---

short LIA_DECL **LIA_set_out_filter**(short hndl, float filter);

Input parameters:

hndl	module handle
filter	new output filter setting in Hz, 0.00524, 0.0131, 0.0262, 0.0524, 0.131, 0.262, 0.524, 1.31, 2.62, 5.24, 13.1, 26.2, or 52.4

Return value:
0 no errors , <0 error code

Description:

The procedure changes output filter setting for the LIA module ‘hndl’. Because the output filter setting is strongly connected to the output rate setting, the output rate can change if a new output filter is set. Therefore, after changing the output filter the output rate parameter should be checked.

The Output Filter setting must be the same for all active LIA modules.

---

short LIA_DECL **LIA_get_out_filter**(short hndl, float far * filter);

Input parameters:

hndl	module handle
filter	pointer to result value

Return value:
0 no errors , <0 error code

Description:

The procedure loads the ‘filter’ variable with the actual value of the output filter frequency for the LIA module ‘hndl’.

---

short LIA_DECL **LIA_set_out_rate**(short hndl, float rate);

Input parameters:

hndl	module handle
rate	new output filter rate setting in samples/sec, 0.02, 0.04, 0.05, 0.08, 0.1, 0.2, 0.4, 0.5, 0.8, 1, 2, 4, 5, 10, 20, 40, 50, 80, 100, or 200

Return value:
0 no errors , <0 error code

Description:

The procedure changes output filter rate setting for the LIA module ‘hndl’. Because the output rate setting is closely connected to the output filter setting, the output filter can change if a new output rate is set. Therefore, after changing the output rate the output filter parameter should be checked.

The Output Filter setting must be the same for all active LIA modules.

---

short LIA_DECL **LIA_get_out_rate**(short hndl, float far * rate);

Input parameters:

hndl	module handle
rate	pointer to result value

Return value:

0 no errors , <0 error code

Description:

The procedure loads the ‘rate’ variable with the actual value of the output filter rate for the LIA module ‘hndl’.

---

short LIA_DECL **LIA_set_offset**(short hndl, short chan, long value);

Input parameters:

hndl	module handle
chan	0 - channel A, 1 - channel B
value	new offset value in range of - 1000000 .. 1000000

Return value:

0 no errors , <0 error code

Description:

The procedure change the offset value of channel ‘chan’ for the LIA module ‘hndl’. Channel offset is used to compensate for offsets in the phase sensitive rectifier and in the ADC of the LIA modules. It is given in ADC steps and is used in the calculation of the ADC samples.

---

short LIA_DECL **LIA_get_offset**(short hndl, short chan, long far * value);

Input parameters:

hndl	module handle
chan	0 - channel A, 1 - channel B
value	pointer to result value

Return value:

0 no errors , <0 error code

Description:

The procedure loads ‘value’ with the actual value of the channel ‘chan’ offset for the LIA module ‘hndl’.

Channel offset is used to compensate for offsets in the phase sensitive rectifier and in the ADC of the LIA modules. It is given in ADC steps and is used in the calculation of the ADC samples.

---

short LIA_DECL **LIA_set_dac**(short hndl, short chan, float value);

Input parameters:

hndl	module handle
chan	0 - DACout1, 1 - DACout2
value	new DAC output value value in the range of - 2.5 .. 2.5 V

Return value:

0 no errors , <0 error code

Description:

The procedure is used to set the output voltage of one of the two uncommited DAC channels (DACout1 or DACout2 at the sub-D connector) on the LIA module ‘hndl’.

---

short LIA_DECL **LIA_get_dac**(short hndl, short chan, float far * value);

Input parameters:

hndl	module handle
chan	0 - DACout1, 1 - DACout2
value	pointer to result value

Return value:

0 no errors , <0 error code

Description:

The procedure loads ‘value’ with the actual output voltage of the DAC ‘chan’ on the LIA module ‘hndl’.

---

short LIA_DECL **LIA_get_eeprom_data**(short hndl, LIA_EEP_Data far *eep_data);

Input parameters:

hndl	module handle
eep_data	pointer to result structure

Return value:

0 no errors , <0 error code

Description:

The structure "eep_data" is filled with the contents of LIA module's EEPROM specified by ‘hndl’. The EEPROM contains production data and adjust parameters of the module. The structure "LIA_EEP_Data" is defined in the file lia_def.h.

Normally, the EEPROM data need not be read explicitly because the EEPROM is read during LIA_init and the module type information and the adjust values are taken into account when the LIA module registers are loaded.

---

short LIA_DECL **LIA_write_eeprom_data**(short hndl, unsigned short write_enable,
LIA_EEP_Data far *eep_data);

Input parameters:

hndl	module handle
write_enable	write enable password
eep_data	pointer to result structure

Return value:

0 no errors , <0 error code

Description:

The function is used to write data to the EEPROM of an LIA module specified by ‘hndl’ by the manufacturer. To prevent corruption of the adjust data by not allowed access the function writes data to the EEPROM only if the ‘write_enable’ password is correct.

---

short LIA_DECL **LIA_get_adjust_parameters**(short hndl, LIA_Adjust_Para far * adjpara);

Input parameters:

hndl	module handle
adjpara	pointer to result structure

Return value:

0 no errors , <0 error code

Description:

The structure 'adjpara' is filled with adjust parameters that are currently in use. The parameters can either be previously loaded from the EEPROM by LIA_init or LIA_get_eeprom_data or - not recommended - set by LIA_set_adust_parameters.

The structure "LIA_Adjust_Para" is defined in the file lia_def.h.

Normally, the adjust parameters need not be read explicitly because the EEPROM is read during LIA_init and the adjust values are taken into account when the LIA module registers are loaded.

---

short LIA_DECL **LIA_set_adjust_parameters**(short hndl, LIA_Adjust_Para far * adjpara);

Input parameters:

hndl	module handle
adjpara	pointer to result structure

Return value:

0 no errors , <0 error code

Description:

The adjust parameters in the internal DLL structures (not in the EEPROM) are set to values from the structure "adjpara". The function is used to set the module adjust parameters to values other than the values from the EEPROM. The new adjust values will be used until the next call of LIA_init. The next call to LIA_init replaces the adjust parameters by the values from the EEPROM. We strongly discourage to use modified adjust parameters, because the module function can be seriously corrupted by wrong adjust values.

The structure "LIA_Adjust_Para" is defined in the file lia_def.h.

---

short LIA_DECL **LIA_start_measure**(short test_trigger);

Input parameters:

test_trigger	0 - do not test sweep trigger, 1 - test sweep trigger

Return value:

0 measurement started, no errors , 1 - waiting for trigger, <0 error code

Description:

LIA_start_measure initialises the measurement with the parameters set before by the LIA_init or other parameter setting functions for all active LIA modules. The procedure initialises the DLL internal variables, performs an auto-calibration routine and enables the readout of the ADC values. If test_trigger is ‘1’, the procedure at the beginning tests whether

a sweep trigger occurred in all active LIA modules. If the trigger is not found in all active LIA modules the procedure returns a ‘1’ without performing other actions.

---

short LIA_DECL **LIA_stop_measure**(void);

Input parameters:

none

Return value:

0

Description:

LIA_stop_measure is used to stop the measurement by a software command. It disables the readout of the ADC values in all LIA modules and resets the internal DLL variables.

---

short LIA_DECL **LIA_read_sample**(short hndl, float far * sampleA, float far *sampleB);

Input parameters:

hndl	module handle
sampleA	pointer to channel A sample
sampleB	pointer to channel B sample

Return value:

0 no errors , <0 error code

Description:

The procedure fills ‘sampleA’ and ‘sampleB’ with the sample values of channel A and B on the LIA module ‘hndl’, but only if a new sample is ready to be read. Please use the LIA_test_if_sample_ready function to check whether a new sample is ready.

---

short LIA_DECL **LIA_test_if_sample_ready** (short far *ready);

Input parameters:

hndl	module handle
ready	pointer to result variable

Return value:

0 no errors , <0 error code

Description:

LIA_test_if_sample_ready is used to check whether a new sample value is available on all active modules. It sets the ‘ready’ variable to 1 if a sample is ready, otherwise ‘ready’ is set to 0.

---

short LIA_DECL **LIA_get_status** (short hndl, short far * status);

Input parameters:

hndl	module handle
status	pointer to result variable

Return value:

0 no errors , <0 error code

Description:

LIA_get_status is used to test the state of LIA module ‘hndl’. The status bits delivered by the function are listed below.

0x1	bit 0 - reference clock present
0x2	bit 1 - input signal overload
0x4	bit 2 - PLL state 0 - Locked , 1 - Unlocked

==========